

Tipos paramétricos

Taller de Álgebra I

Segundo cuatrimestre de 2015

Repaso tipos enumerados

Logo

```
data Direccion = Norte | Sur | Este | Oeste
type Tortuga = (Pos, Direccion)
type Pos = (Integer, Integer)
```

Ejercicios

- 1 arranca :: Tortuga que devuelva una tortuga en el (0,0) mirando hacia el Norte
- 2 girarDerecha :: Tortuga -> Tortuga que gire la tortuga 90 grados a la derecha sin moverla de lugar.
- 3 avanzar :: Tortuga -> Integer -> Tortuga hace avanzar hacia el lugar donde mira la tortuga la distancia indicada.

Tipos algebraicos paramétricos: Figuras

Los constructores de tipos algebraicos pueden tomar parámetros, veamos un ejemplo:

```
data Figura = Rectangulo Float Float Float Float
            | Circulo Float Float Float
```

Ejercicios, implementar las siguientes funciones

- 1 `c1 :: Figura` que devuelve un círculo que se encuentra en el origen con radio `pi`.
- 2 `r1 :: Float -> Figura` que toma 1 parámetro: `x` y luego retorna un rectángulo que tiene un punto en el origen y el otro a distancia `x`.
- 3 `area :: Figura -> Float` que devuelva el área de la figura

Supongamos que ahora cambiamos las definiciones de la siguiente manera:

Más declarativos!

```
data Punto = Point Float Float
data Figura2 = Rectangulo2 Punto Punto | Circulo2 Punto Float
```

- 1 Reimplementar la función `area :: Figura2 -> Float`

Una *progresión aritmética* es el conjunto vacío o un conjunto infinito de enteros tal que entre dos elementos consecutivos del conjunto hay siempre la misma diferencia.

Ejemplos:

- ▶ $\{\dots, -2, 1, 4, 7, 10, \dots\}$,
- ▶ $\{\dots, -18, -9, 0, 9, 18, 27, \dots\}$,
- ▶ \emptyset .

Se puede describir una progresión aritmética no vacía como todos los enteros congruentes a un cierto número, módulo otro.

Ejemplos:

- ▶ $\{\dots, -2, 1, 4, 7, 10, \dots\} = \{a \in \mathbb{Z} \mid a \equiv 1 \pmod{3}\}$,
- ▶ $\{\dots, -18, -9, 0, 9, 18, 27, \dots\} = \{a \in \mathbb{Z} \mid a \equiv 0 \pmod{9}\}$.

Para hacer

```
data ProgAritmetica = Vacio | CongruentesA Integer Integer
```

Es decir,

$\{a \in \mathbb{Z} \mid a \equiv 1 \pmod{3}\}$ es CongruentesA 1 3,

$\{a \in \mathbb{Z} \mid a \equiv 0 \pmod{9}\}$ es CongruentesA 0 9.

Ejercicios

- ▶ `esMultiplo :: Integer -> Integer -> Bool`
`esMultiplo 9 3 ~> True`
- ▶ `pertenece :: Integer -> ProgAritmetica -> Bool`
`pertenece 13 Vacio ~> False`
`pertenece 13 (CongruentesA 5 4) ~> True`
- ▶ `incluido :: ProgAritmetica -> ProgAritmetica -> Bool`
`incluido (CongruentesA 4 6) (CongruentesA 10 3) ~> True`

Alto show

Muy lindo, pero

```
Prelude> CongruentesA 3 8
```

```
<interactive>:70:1:
```

```
  No instance for (Show ProgAritmetica)
```

```
    arising from a use of 'print'
```

```
  In a stmt of an interactive GHCi command: print it
```

Esto se debe a que *Haskell* no sabe mostrar en pantalla una *ProgAritmetica*.

Solución 1: deriving (Haskell decide qué mostrar)

```
data ProgAritmetica = Vacio | CongruentesA Integer Integer
  deriving Show
```

```
Prelude> CongruentesA 3 9
```

```
CongruentesA 3 9
```

```
Prelude> Vacio
```

```
Vacio
```

Alto show

Solución 2: le decimos a Haskell cómo hacer `show` de una `ProgAritmetica`:

```
data ProgAritmetica = Vacio | CongruentesA Integer Integer
instance Show ProgAritmetica where
    show Vacio = "Vacio"
    show (CongruentesA x d) = "Una progresion no vacia"
```

```
Prelude> CongruentesA 3 9
Una progresion no vacia
```

```
Prelude> CongruentesA 4 7
Una progresion no vacia
```

```
Prelude> Vacio
Vacio
```

Alto show: Mejoremos lo que se muestra

Implementar la función `show` de manera que las progresiones se muestren de la siguiente manera:

```
Prelude> CongruentesA 3 8
{a en Z | a = 3 (mod 8)}
```

```
Prelude> Vacio
{}
```

Ejercicios: Implementar las siguientes funciones

- ▶ $\text{suma } p \ q = \{(a + b) \mid a \in p, \ b \in q\}$.
Hacer suma tal que
`suma (CongruentesA 3 6) (CongruentesA 2 4) ~> CongruentesA 5 2.`
- ▶ Hacer `interseccion :: ProgAritmetica -> ProgAritmetica -> ProgAritmetica`
- ▶ Programar `iguales :: ProgAritmetica -> ProgAritmetica -> Bool`
`iguales (CongruentesA 22 5) (CongruentesA 2 5) ~> True`
- ▶ Hacer que las progresiones sean instancias de `Eq` utilizando la igualdad programada anteriormente.
Sugerencia, buscar en <http://aprendehaskell.es/content/ClasesDeTipos.html> la definición de `TrafficLight`
- ▶ Hacer `tieneSolucion :: Integer -> ProgAritmetica -> Bool` que diga si una ecuación de congruencia tiene solución. Explícitamente,
`tieneSolucion t (CongruentesA a m) ~> True`
si y solo si la ecuación
 $tx \equiv a \pmod{m}$ tiene solución **esperar para resolverlo si no lo vieron en la teórica**